A modeling environment for reified temporal-causal network models

Treur, Jan

published in
Network-Oriented Modeling for Adaptive Networks
2020

DOI (link to publisher)
10.1007/978-3-030-31445-3_9

document version
Publisher's PDF, also known as Version of record

Chapter 9
A Modeling Environment for Reified Temporal-Causal Network Models

Abstract The introduced multilevel reified (temporal-causal) network architecture is the basis of the implementation of a dedicated software environment developed by the author in Matlab. The environment includes a combination function library and a generic computational reified network engine. It uses role matrices specifying the characteristics for the designed network model as input. Based on this input, the computational reified network engine can be used to generate simulations for the network model, thereby using combination functions from the library. In this chapter, this software environment is described in more detail.

9.1 Introduction

The notion of temporal-causal network model is a point of departure (Treur 2016, 2019a). Based on this notion, in Chaps. 3 and 4 the architecture of a reified network model has been introduced and illustrated by, respectively

- a first-order adaptive example Social Network with adaptive responding speed and adaptive aggregation for contagion
- a second-order adaptive example Mental Network showing plasticity and metaplasticity.

See also (Treur 2018a, b). In the subsequent Chaps. 5–8, five more applications of this architecture have been shown, respectively,
for

- a second-order adaptive Mental Network for adaptive decision making based on a state-connection adaptation principle
- a second-order adaptive Social Network for Bonding by Homophily
- a fourth-order adaptive Biological Network for evolutionary processes in handling pathogens
- a third-order adaptive Mental Network based on a Strange Loop, and a first-order mutually reified Mental Network for adaptive decision making based on a Strange Loop.

J. Treur, *Network-Oriented Modeling for Adaptive Networks: Designing Higher-Order Adaptive Biological, Mental and Social Network Models*, Studies in Systems, Decision and Control 251, https://doi.org/10.1007/978-3-030-31445-3_9

To fully exploit the potential of this approach for such applications for adaptive networks, a dedicated modeling environment has been developed. This modeling environment uses a specific implementation-independent modeling format for a reified network architecture based on role matrices (see Chaps. 2–4, and many later examples in the book). This format has a basis in the implementation-independent mathematical notions of matrix and function. The notion of role matrix provides an alternative and more compact specification format for networks, in contrast to connection matrices that are often used as main means to specify networks at an implementation-independent level. Section 9.2 briefly summarizes role matrices as specification format for reified adaptive temporal-causal network models. This role matrix format can be used for implementation in different ways; one example implementation is the computational reified temporal-causal network engine discussed here.

The basic elements of the specification format are network characteristics that are declarative by nature: connection weights, combination functions, speed factors, grouped in role matrices. Together these elements define in a standard manner a set of first-order difference or differential equations (as shown in Chap. 2, Tables 2.1 and 2.2), which are used computationally in the software environment. The model's behavior is fully determined and explainable by these declarative temporal specifications (given some initial values). The modeling process is strongly supported by using these declarative building blocks. By using a reified network architecture, the scope is substantially extended to adaptive networks of any order. As shown in earlier chapters, very complex adaptive patterns can be modeled easily in temporal declarative form. The universal combination function and universal difference equation as described in Chap. 3, Sect. 3.5 (see also Chap. 10) form a basis to address multiple orders of adaptation.

Combination functions play a central role in the modeling approach. The software environment includes a combination function library (currently with 35 functions), which is discussed in Sect. 9.3. In Sect. 9.4 the computational reified temporal-causal network engine is described that has been developed. This is the software that takes the declarative specification of Sect. 9.2 in terms of role matrices and makes it run. The design of this software is structure-preserving in relation to the mathematical description and can be easily translated into any other software environment that supports the mathematical notions of function and matrix.

9.2 Role Matrices as a Specification Format for Reified Network Models

For networks, matrices are often considered a way that is easily accessible computationally and vastly used in network modeling and analysis. The choice was made to use specific types of matrices (called *role matrices*) to define a specification format to design reified network models, and as a well-defined basis for implementation. The implementation described here is based on Matlab ('Matrix

Laboratory') which handles matrices well; but any other software environment supporting matrices can be used as well. Role matrices were introduced in Chap. 2 as matrices grouping the relevant values for the characteristics of the network structure, for example as used in a given simulation scenario. For a nonadaptive network, role matrices are indeed just neatly structuring these data in a table format (e.g., in Word or in Excel) that provides a complete description of the design of the network model. When these role matrices are copied to Matlab, the network model can be executed (see Sect. 9.4). In Chap. 3 the use of role matrices was extended to reified networks. In that case, some of the cells of these matrices indicate adaptive network structure characteristics, which means that they do not contain a constant value, but a variable value instead; these cells were shaded in red, whereas the value cells were in green. This 'variable value' is indicated in that cell by putting the name of the reification state that plays the role for that characteristic. This indication specifies the standard downward causal connection for that reification state. The specific role matrix in which this reification state name is indicated defines for which role it is a reification state; see also Table 9.1.

More specifically, the role matrices have rows for all states of the network. For an example, borrowed from Chap. 4, see Box 9.1; for the picture of the model, see Chap. 4, Sect. 4.4.1, Fig. 4.3. In the row for state Y, in base role matrix **mb** for *base connectivity* it is specified which other states have an impact on Y. This corresponds to the upward and horizontal incoming arrows for state Y in a conceptual graphical representation of the model. In the other (non-base) role matrices, other roles are

Table 9.1 Downward causal connections and role matrices: how and where specify what

In conceptual graphical representation of the model	State name	State number	Role	In role matrix
Downward arrow from a reification state for an adaptive connection weight from state X to state Y	$\mathbf{W}_{X,Y}$	X_i	Connection weight reification state for $\boldsymbol{\omega}_{X,Y}$	**mcw** as notation X_i in the cell for the weight $\boldsymbol{\omega}_{X,Y}$ of the connection from X to Y
Downward arrow from a reification state for an adaptive speed factor for state Y	$\mathbf{H}_Y$	X_j	Speed factor reification state for $\boldsymbol{\eta}_Y$	**ms** as notation X_j in the cell for the value $\boldsymbol{\eta}_Y$ of the speed factor of Y
Downward arrow from a reification state for an adaptive combination function weight for state Y	$\mathbf{C}_{i,Y}$	X_k	Combination function weight reification state for weight $\boldsymbol{\gamma}_{i,Y}$	**mcfw** as notation X_k in the cell for the weight $\boldsymbol{\gamma}_{i,Y}$ of combination function i for state Y
Downward arrow from a reification state for an adaptive combination function parameter for state Y	$\mathbf{P}_{i,j,Y}$	X_l	Combination function parameter reification state for parameter $\boldsymbol{\pi}_{i,,j,Y}$	**mcfp** as notation X_l in the cell for the value $\boldsymbol{\pi}_{i,,j,Y}$ of parameter j of combination function i for state Y

indicated (either values or reification states playing that role): in **mcw** *connection weights*, in **ms** *speed factors,* in **mcfw** *combination function weights*, and in **mcfp** *combination function parameters*. In the case of reification states, these (non-base) role indications correspond to downward arrows in a conceptual graphical representation of the model. Each of these downward arrows in such a picture gets a special effect assigned due to the role matrix in which it is indicated: the connected reification state plays the role of connection weight, the role of speed factor, the role of combination function weight, or the role of a combination function parameter. The special causal effect of a reification state on the related base state Y has to be according to this role. For example, the value of a reification state with role speed factor has to be used for the speed factor characteristic in the processing for Y, and not for anything else. See Table 9.1 for an overview of these non-base roles.

Box 9.1 Example specification of a second order reified network model for plasticity and metaplasticity by role matrices

mb	base connectivity	1	2	3	4
X_1	ss_s	X_1			
X_2	srs_s	X_1			
X_3	bs_s	X_2			
X_4	ps_a	X_2	X_3		
X_5	$\mathbf{W}srs_s,ps_a$	X_2	X_4	X_5	
X_6	$\mathbf{T}srs_s$	X_2	X_4	X_5	X_6
X_7	$\mathbf{T}ps_a$	X_2	X_4	X_5	X_7
X_8	$\mathbf{HW}srs_s,ps_a$	X_2	X_4	X_5	X_8
X_9	$\mathbf{MW}srs_s,ps_a$	X_2	X_4	X_5	X_9

mcw connection weights		1	2	3	4
X_1	ss_s	1			
X_2	srs_s	1			
X_3	bs_s	1			
X_4	ps_a	X_5	1		
X_5	$\mathbf{W}srs_s,ps_a$	1	1	1	
X_6	$\mathbf{T}srs_s$	-0.4	-0.4	1	1
X_7	$\mathbf{T}ps_a$	-0.4	-0.4	1	1
X_8	$\mathbf{HW}srs_s,ps_a$	1	1	-0.1	1
X_9	$\mathbf{MW}srs_s,ps_a$	1	1	1	1

mcfw combination function weights		1	2	3
		eucl	alogistic	hebb
X_1	ss_s	1		
X_2	srs_s		1	
X_3	bs_s		1	
X_4	ps_a		1	
X_5	$\mathbf{W}srs_s,ps_a$			1
X_6	$\mathbf{T}srs_s$		1	
X_7	$\mathbf{T}ps_a$		1	
X_8	$\mathbf{HW}srs_s,ps_a$		1	
X_9	$\mathbf{MW}srs_s,ps_a$		1	

mcfp	function	1		2		3	
		eucl		alogistic		hebb	
	parameter	1 n	2 λ	1 σ	2 τ	1 μ	2
X_1	ss_s	1	1				
X_2	srs_s			5	X_6		
X_3	bs_s			5	0.2		
X_4	ps_a			5	X_7		
X_5	$\mathbf{W}srs_s,ps_a$					X_9	
X_6	$\mathbf{T}srs_s$			5	0.7		
X_7	$\mathbf{T}ps_a$			5	0.7		
X_8	$\mathbf{HW}srs_s,ps_a$			5	1		
X_9	$\mathbf{MW}srs_s,ps_a$			5	1		

ms speed factors		1
X_1	ss_s	0.5
X_2	srs_s	0.5
X_3	bs_s	0.2
X_4	ps_a	0.5
X_5	$\mathbf{W}srs_s,ps_a$	X_8
X_6	$\mathbf{T}srs_s$	0.3
X_7	$\mathbf{T}ps_a$	0.3
X_8	$\mathbf{HW}srs_s,ps_a$	0.5
X_9	$\mathbf{MW}srs_s,ps_a$	0.1

Table 9.2 Role matrices **mcwa** and **mcwv** for the connection weights for the specification in Box 9.1

mcwa		1	2	3	4
X_1	ss_s				
X_2	srs_s				
X_3	bs_s				
X_4	ps_a	5			
X_5	$\mathbf{W}_{srs_s,ps_a}$				
X_6	$\mathbf{T}_{srs_s}$				
X_7	$\mathbf{T}_{ps_a}$				
X_8	$\mathbf{Hw}_{srs_s,ps_a}$				
X_9	$\mathbf{Mw}_{srs_s,ps_a}$				

mcwv		1	2	3	4
X_1	ss_s	1			
X_2	srs_s	1			
X_3	bs_s	1			
X_4	ps_a		1		
X_5	$\mathbf{W}_{srs_s,ps_a}$	1	1	1	
X_6	$\mathbf{T}_{srs_s}$	-0.4	-0.4	1	
X_7	$\mathbf{T}_{ps_a}$	-0.4	-0.4	1	
X_8	$\mathbf{Hw}_{srs_s,ps_a}$	1	1	-0.1	1
X_9	$\mathbf{Mw}_{srs_s,ps_a}$	1	1	1	1

Moreover, notice that the connected reification states are given standard names to reflect their role **H**, **W**, **C** or **P**, meant for human understanding, and these are not used in the execution of the model. State names used in the implementation are just the X_1, X_2, … based on the numbering of the states; see also Table 9.1. In the execution of the model, the role matrices fully define which are the downward causal connections and which of the network structure characteristics for target state Y they affect as their special effect.

Note that the role matrices used in the reified network model specification format have a compact format, and also specify an ordering, which is important as combination functions used to integrate the impact from multiple connections which are not always symmetric in their arguments.

Here, the red cells represent the downward causal connections from the reification states in pictures as shown in Chap. 4, Fig. 4.3, with their specific roles **W**, **H**, **C**, **P** indicated by the type of role matrix in which such a red cell occurs. The static values are in the green cells, and the remaining cells are empty. According to this red/green/empty partition, each role matrix can be split into two matrices: one for the green part (and the other cells empty) and one for the red part (and the other cells empty). The former matrix is called the *adaptation matrix* (with an **a** added to the name) and the latter the *values matrix* (with a **v** added to the name); for example, see Table 9.2. This will be discussed in more detail in Sect. 9.4.

9.3 The Combination Function Library

The combination function library currently consists of 35 functions. Three main groups are briefly discussed first, after which the standard format for these functions is discussed. More details can be found at Treur (2019b) and at https://www.researchgate.net/publication/336681331.

9.3.1 Different Classes of Combination Functions

As a first class, the library currently includes the following functions that in principle are suitable to model social contagion in social networks, but can also be used in other network models (for further analysis of these functions and the emerging behaviour entailed by them, see also Chap. 11, Sects. 11.4–11.6):

```
eucl(p,v), alogistic(p,v), slogistic(p,v), invtan(p,v),
smin(p,v),smax(p,v), aminmax(p,v), sgeomean(p,v)
```

Here, and below, `p` is the vector of parameters and `v` the vector of values for the function. As a second class, to model reification states for Hebbian learning, currently available options are:

```
hebb(p,v), sconnhebb(p,v), srconnhebb(p,v),
srstateshebb(p,v), sstateshebb(p,v)
```

These functions are analysed further in Chap. 14, Sect. 14.6, and satisfy the relevant properties of Hebbian learning combination functions discussed in Chap. 14, Sect. 14.4.1. Similarly, another class of functions is available to model reification states for bonding by homophily (for analysis of these functions, see also Chap. 13):

```
slhomo(p,v), sqhomo(p,v), alhomo(p,v), aqhomo(p,v),
cubehomo(p,v), exphomo(p,v), log1homo(p,v), log2homo(p,v),
sinhomo(p,v), tanhomo(p,v), multicriteriahomo(p,v)
```

These functions are shown in particular in Chap. 13, Sect. 13.3.2 and satisfy the relevant properties of bonding by homophily combination functions discussed in Chap. 13, Sect. 13.5.

Note that when more than one combination function is selected for one given state with nonzero combination function weights in matrix **mcfw**, then these functions should use the same shared sequence of values. This is automatically the case with functions of the first group, which are symmetric in their arguments and have a variable number of k arguments. For functions from the other groups this not automatically the case, as they are not symmetric in their arguments; then auxiliary variables may have to be added to be able to use the same sequence of values for multiple functions, from which each function only actually uses the subsequence that is relevant for it. For an illustration of how this is done, see Chap. 5, Sect. 5.4.1
.

9.3.2 The Standard Format of Combination Functions

A complete specification of these combination functions can be found at (Treur 2019b). To obtain a general format easily usable within the software environment, these functions were numbered and rewritten in the standard *basic combination function* form

`bcf(i, p, v)`

where `i` is the number of the basic function, **p** is its vector of parameters and **v** is a vector of values. This was implemented in Matlab (and stored as `bcf.m`) by the definition for `bcf` shown in Box 9.2.

Box 9.2 Specification of `bcf` getting the basic combination functions in a standard format.

```
function x = bcf(i,p,v)
% bcf =  basic combination functions; this function combi-
nes all basic
combination functions in one format
if    i ==1       x = eucl(p,v);
elseif i ==2      x = alogistic(p,v);
elseif i ==3      x = hebb(p,v);
elseif i ==4      x = scm(p,v);
elseif i ==5      x = slhomo(p,v);
elseif i ==6      x = sqhomo(p,v);
elseif i ==7      x = alhomo(p,v);
elseif i ==8      x = aqhomo(p,v);
elseif i ==9      x = sconnhebb(p,v);
elseif i ==10     x = srconnhebb(p,v);
elseif i ==11     x = srstateshebb(p,v);
elseif i ==12     x = sstateshebb(p,v);
elseif i ==13     x = slogistic(p,v);
elseif i ==14     x = cubehomo(p,v);
elseif i ==15     x = exphomo(p,v);
elseif i ==16     x = log1homo(p,v);
elseif i ==17     x = log2homo(p,v);
elseif i ==18     x = sinhomo(p,v);
elseif i ==19     x = tanhomo(p,v);
elseif i ==20     x = invtan(p,v);
elseif i ==21     x = id(p,v);
elseif i ==22     x = complementid(p,v);
elseif i ==23     x = product(p,v);
```

```
elseif i ==24    x = coproduct(p,v);
elseif i ==25    x = sminimum(p,v);
elseif i ==26    x = smaximum(p,v);
elseif i ==27    x = aproduct(p,v);
elseif i ==28    x = aminmax(p,v);
elseif i ==29    x = multicriteriahomo(p,v);
elseif i ==30    x = ssum(p,v);
elseif i ==31    x = adnormsum(p,v);
elseif i ==32    x = adnormeucl(p,v);
elseif i ==33    x = sgeomean(p,v);
elseif i ==34    x = stepmod(p,v);
elseif i ==35    x = stepmodopp(p,v);
end
end
```

The selection of a sequence of combination functions from the library for a given network model is indicated by

mcf = [1 2 3]

where the sequence 1, 2, 3 (referring to the numbering of the library specified by bcf in Box 9.2) can be replaced by any sequence of any length of numbers from 1 to 35. After this specification, for the specific network model the numbering from this subsequence is used. For example, if in a network model specification **mcf** = [5 9 12 18] is indicated, then for that network model the 5th function from the library (**slhomo**) is indicated as basic combination function number 1, the 9th (**sconnhebb**) as number 2, and so on. It should be kept in mind that in principle this (local) numbering is different for each network model.

9.4 The Computational Reified Temporal-Causal Network Engine

The designed computational reified network engine takes a specification in the role matrices format as described in Sect. 9.2 and generates simulations for the model. It uses two main steps: first retrieving the characteristics of the network model from the role matrices (Sect. 9.4.2), and next performing the actual execution (Sect. 9.4.3). But before that, the role matrices have to be entered (Sect. 9.4.1).

9.4.1 Splitting the Role Matrices and Copying Them to Matlab

First, each role matrix (which can be specified easily by a table in Word or in Excel, for example) is copied or read in Matlab in two variants:

- a *value matrix* for the static values (adding the letter **v** to the name) from the green cells, and
- an *adaptivity matrix* for the adaptive values represented by reification states (adding the letter **a** to the name) from the red cells, thereby replacing X_j by the index j.

Note that states X_j are represented in Matlab by their index number j. For example, from **mcw** two matrices **mcwa** (adaptive connection weights matrix) and **mcwv** (connection weight values matrix) are derived in this way (see Table 9.2). The (index) numbers in **mcwa** indicate the state numbers of the reification states where the values can be found (so, the 5 in **mcwa** should be interpreted as state X_5, not as a value 5), and in **mcwv** the numbers indicate the static values directly.

Before copying, the empty cells are filled for Matlab with NaN (Not a Number) indications, to get a neat rectangular matrix structure. After copying to Matlab this results in the matrices in Matlab representation as shown in Table 9.3.

Note that for a given network model the values in the value matrices can differ per scenario addressed, they represent the specific settings for the simulation scenario. As another example, for the 3D role matrices **mcfpa** and **mcfpv** their representations in Matlab are as depicted in Table 9.4.

9.4.2 Retrieving Information from the Role Matrices

During execution of a simulation, for each step from k to $k + 1$ (with step size Δt, in Matlab denoted by `dt`), above role matrices are used. As a first step, for each state X_j the right values (either the fixed value, or the adaptive value found in the indicated reification state) are assigned to:

Table 9.3 Role matrices **mcwa** and **mcwv** for connection weights as represented within Matlab

```
mcwa = [NaN NaN NaN NaN
NaN NaN NaN NaN
NaN NaN NaN NaN
5   NaN NaN NaN
NaN NaN NaN NaN
NaN NaN NaN NaN
NaN NaN NaN NaN
NaN NaN NaN NaN
NaN NaN NaN NaN
]
```

```
mcwv = [1   NaN   NaN   NaN
1     NaN   NaN   NaN
1     NaN   NaN   NaN
NaN   1     NaN   NaN
1     1     1     NaN
0     -0.4  1     1
0     -0.4  1     1
0     1    -0.4   1
0     1     1     1
]
```

Table 9.4 Role matrices **mcfpa** and **mcfpv** for combination function parameters as represented in Matlab

mcfpa = cat(3,[NaN NaN	**mcfpv** = cat(3,[1 1
NaN NaN	NaN NaN
NaN NaN	NaN NaN
NaN NaN	NaN NaN
NaN NaN	NaN NaN
NaN NaN	NaN NaN
NaN NaN	NaN NaN
NaN NaN	NaN NaN
NaN NaN	NaN NaN
],[NaN NaN	],[NaN NaN
NaN 6	5 NaN
NaN NaN	5 0.2
NaN 7	5 NaN
NaN NaN	NaN NaN
NaN NaN	5 0.7
NaN NaN	5 0.7
NaN NaN	5 1
NaN NaN	5 1
],[NaN NaN	],[NaN NaN
NaN NaN	NaN NaN
NaN NaN	NaN NaN
NaN NaN	NaN NaN
9 NaN	NaN NaN
NaN NaN	NaN NaN
NaN NaN	NaN NaN
NaN NaN	NaN NaN
NaN NaN	NaN NaN
])	])

These are basically three 2D matrices each with a first (vertical) dimension for the states and a second (horizontal) dimension for the parameters (2 in this case), indicated in Box 9.1 in **mcfpv** as three subboxes for eucl, alogistic, and hebb, respectively. These 2D matrices are grouped in the vertical direction according to the third dimension for the 3 combination functions selected for this specific model (**eucl(..)**, **alogistic(..)**, and **hebb(..)**, respectively), which results in the following 3D matrix representation in Matlab

`s(j, k)`	speed factor for state X_j
`b(j, p, k)`	value for the pth state with outgoing base connection to state X_j
`cw(j, p, k)`	connection weight for the pth state with outgoing base connection to state X_j
`cfw(j, m, k)`	weight for the mth combination function for X_j
`cfp(j, p, m, k)`	the pth parameter value of the mth combination function for X_j.

This is done by the code shown in Box 9.3.

Box 9.3 First part of the Reified Temporal-Causal Network Engine developed in Matlab; see also Treur (2019b)

```
if not(isnan(msa(j, 1)))
       s(j, k) = X(msa(j, 1), k);
     elseif not(isnan(msv(j, 1)))
       s(j, k) = msv(j, 1);
     elseif isnan(msv(j, 1))
           s(j, k) = 0;
end
     %  This extracts the speed factor value from the
appropriate role matrix msa or msv;
     %  if none of them gives a value, then default value 0 is as-
signed
for p = 1:1:size(mb,2)
     if not(isnan(mb(j, p)))
           b(j, p, k) = X(mb(j,p), k);
     elseif isnan(mb(j, p))
           b(j, p, k) = 0;
     end
end
     %  This extracts the relevant base state values
from the appropriate role matrix mb;
     %  if none of them gives a value, then default value 0
is assigned
for p = 1:1:size(mcwa,2)
     if not(isnan(mcwa(j, p)))
           cw(j, p, k) = X(mcwa(j,p), k);
     elseif not(isnan(mcwv(j, p)))
           cw(j, p, k) = mcwv(j, p);
     elseif isnan(mcwv(j, p))
           cw(j, p, k) = 0;
     end
end
     %  This extracts the connection weights from the
appropriate role matrix mcwa or mcwv;
     %  if none of them gives a value, then default value 0
is assigned
for m = 1:1:size(mcfwa,2)
     if not(isnan(mcfwa(j, m)))
           cfw(j, m, k) = X(mcfwa(j, m), k);
```

```
     elseif not(isnan(mcfwv(j, m)))
            cfw(j, m, k) = mcfwv(j, m);
     elseif isnan(mcfwv(j, m))
            cfw(j, m, k) = 0;
     end
end
     % This extracts the combination function weights from the
appropriate role
     % matrix mcfwa or mcfwv;
     % if none of them gives a value, then default value 0
is assigned
for p = 1:1:nocfp
     for m = 1:1:nocf
              if not(isnan(mcfpa(j, p, m)))
                     cfp(j, p, m, k) = X(mcfpa(j, p, m), k);
              elseif not(isnan(mcfpv(j, p, m)))
                     cfp(j, p, m, k) = mcfpv(j, p, m);
     elseif isnan(mcfpv(j, p, m))
                     cfp(j, p, m, k) = 1;
              end
     end
end
     % This extracts the combination function parameter values
from the appropriate
     % role matrix mcfpa or mcfpv;
     % if none of them gives a value, then default value 1
is assigned
```

9.4.3 *The Iteration Step from* t *to* t + *Δ*t

Then, as a second part of the computational reified network engine, for the step from k to $k + 1$ the following is applied for each j; here `X(j,k)` denotes $X_j(t)$ for t = `t(k) = k*dt`; see Box 9.4.

Box 9.4 Second part of the Reified Temporal-Causal Network Engine developed in Matlab; see also Treur (2019b)

```
for m = 1:1:nocf
cfv(j,m,k) =
  bcf(mcf(m), squeeze(cfp(j, :, m, k)), squeeze(cw(j, :, k)).
*squeeze(b(j, :, k)));
end
     %  This calculates the combination function values cfv(j,
m,k) for each
     %  combination function mcf(m) for state j at k
aggimpact(j, k) = dot(cfw(j, :, k), cfv(j, :, k))/sum(cfw
(j, :, k));
     %  The aggregated impact for state j at k as the inproduct
of combination
     %  function weights and combination function values,
scaled by the sum of these
     %  weights
X(j,k + 1) = X(j,k) + s(j,k)*(aggimpact(j,k) - X(j,k))*dt;
     %  The iteration step from k to k + 1 for state j
t(k + 1) = t(k) + dt;
     %  Keeping track of time
```

As can be seen in Box 9.4, the structure of the code of this computational reified network engine is quite compact; the essential computational core is only 5 lines of code! This is possible because (in contrast to the hybrid approach discussed in Chap. 1, Sect. 1.4.1) a unified approach is applied to all levels of reification. This is enabled by the universal difference equation discussed in Chap. 10, and resembles the formulae in Chap. 4, Table 4.1: structure-preserving implementation.

Note that functions with multiple groups of arguments in Matlab get vector arguments where groups of arguments become vectors of variable length. For example, the basic combination function expression $\text{bcf}_i(P_{1,i}, P_{2,i}, W_1V_1, \ldots, W_kV_k)$ as part of the universal difference equation defined in Chap. 4, Sect. 4.3.2 becomes `bcf(i, p, v)` in Matlab with vectors p = $[P_{1,i}, P_{2,i}]$ for function parameters and v = $[W_1V_1, \ldots, W_kV_k]$ for the values of the function arguments. This format `bcf (i, p, v)` is also used as the basis of the combination function library developed (currently numbered by i = 1 to 35); for an overview of this basic combination function library, see Treur (2019b).

9.5 Discussion

The dedicated modeling environment as described in this chapter includes a dedicated specification format for reified networks and comes with an implemented dedicated computational reified network engine, which can simply run such specifications. Moreover, a library consisting currently of 35 combination functions is offered, which can be extended easily. Using this software environment, the design process of a network model can be focused in a declarative manner on the reified network specification that serves as a design description. All kinds of complex (higher order) adaptive dynamics are covered without being bothered by implementation details. This provides a very powerful modeling environment, which is easy to use and really compact. It uses the reified network architecture, and handles all orders of adaptation in a unified manner, using the universal difference equation.

Note that in the software as described at every iteration step from t to $t + \Delta t$ retrieval of information from the role matrices takes place. An alternative option is to apply a form of compilation before running a simulation so that retrieval of information from the role matrices takes place once and not all the time. Such an alternative implementation will be described in Chap. 10, Sect. 10.7. In that case, during the compilation step, the universal difference equation is instantiated for each state by the information retrieved from the role matrices. This results in a collection of different instantiated difference equations for all states that can be run by any general purpose difference equation simulator. It has not been tested yet, but this perhaps could provide higher efficiency at simulation time, although also compilation time has to be added in that case; this is beyond the scope of the current book. More details can be found in Chap. 10, Sect. 10.

References

Treur, J.: Network-Oriented Modeling: Addressing Complexity of Cognitive, Affective and Social Interactions. Springer Publishers (2016)

Treur, J.: Network reification as a unified approach to represent network adaptation principles within a network. In: Proceedings of the 7th International Conference on Natural Computing. Lecture Notes in Computer Science, vol 11324, pp. 344–358. Springer Publishers (2018a)

Treur, J.: Multilevel network reification: representing higher-order adaptivity in a network. In: Proceedings of the 7th International Conference on Complex Networks and their Applications, Complex Networks' 18, vol. 1. Studies in Computational Intelligence, vol. 812, pp. 635–651. Springer (2018b)

Treur, J.: The ins and outs of network-oriented modeling: from biological networks and mental networks to social networks and beyond. In: Transactions on Computational Collective Intelligence, vol. 32, pp. 120–139. Springer Publishers, Contents of Keynote Lecture at ICCCI'18 (2019a)

Treur, J.: Design of a Software Architecture for Multilevel Reified Temporal-Causal Networks (2019b). Doi: https://doi.org/10.13140/rg.2.2.23492.07045. Url: https://www.researchgate.net/publication/333662169